НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

(повненайменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»

В.о. завідувача кафедри

(підпис) _____ (ініціали, прізвище) *Олександр ПАВЛОВ*

“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Програмне забезпечення для трансляції Nib-файлів у SwiftUIкод*

Виконав: студент IV курсу, групи *ІП-63 Віхляєв Олександр Олександрович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ст.викл. кафедри АСОІУ Халус О.А.* _____
посада, науковийступінь, вченезвання, прізвище, і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада, науковийступінь, вченезвання, прізвище, і ім’я, по батькові (підпис)

Рецензент:

проф. каф. ТК, д.т.н., проф. Стенін О.А.
посада, науковийступінь, вченезвання, прізвище, ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2020 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Віхляєва Олександра Олександровача
(прізвище, ім’я, по батькові)

1. Тема проєкту « Програмне забезпечення для трансляції Nib-файлів у SwiftUI код »

керівник проєкту *Халус О.А., ст. викладач*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту «08» *червня 2020 року* **3. Вихідні дані до проєкту**

Технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог, математичне забезпечення

2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних

3) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, процесів тестування, опис контрольного прикладу

4) Впровадження та супровід програмного забезпечення: розгортання програмного

забезпечення, робота з програмним забезпеченням

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна класів програмного забезпечення

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*19.03.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*26.03.2020*	
3.	*Постановка та формалізація задачі*	*26.03.2020*	
4.	*Аналіз вимог до програмного забезпечення*	*02.04.2020*	
5.	*Алгоритмізація задачі*	*02.04.2020*	
6.	*Моделювання програмного забезпечення*	*09.04.2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*16.04.2020*	
8.	*Розробка архітектури програмного забезпечення*	*23.04.2020*	
9.	*Розробка програмного забезпечення*	*30.04.2020*	
10.	*Налагодження програми*	*07.05.2020*	
11.	*Виконання графічних документів*	*14.05.2020*	
12.	*Оформлення пояснювальної записки*	*21.05.2020*	
13.	*Подання ДП на попередній захист*	*28.05.2020*	
14.	*Подання ДП рецензенту*	*03.05.2020*	
15.	*Подання ДП на основний захист*	*08.06.2020*	

Студент _____ Олександр ВІХЛЯЄВ
(підпис)

Керівник _____ Олена ХАЛУС
(підпис)

№з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-6307.045490.02.81	Пояснювальна записка	50	
3	А4	КПІ.ІП-6307.045490.03.91	Технічне завдання	12	
4	А4	КПІ.ІП-6307.045490.04.51	Програма та методика тестування	12	
5	А4	КПІ.ІП-6307.045490.05.34	Керівництво користувача	4	
6	А4	КПІ.ІП-6307.045490.06.13	Опис програми	14	
7	А3	КПІ.ІП-6307.045490.07.99	Схема структурна варіантів використання	1	
8	А3	КПІ.ІП-6307.045490.07.99	Схема структурна класів програмного забезпечення	1	
9	А3	КПІ.ІП-6307.045490.07.99	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6307.045490.00.90		
Розробн.		Віхляєв О.О.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Халус О.А.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-63	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: Програмне забезпечення для трансляції Nib-файлів у SwiftUI код ___
_____
_____

Київ – 2020 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з чотирьох розділів, містить 14 таблиць, 3 додатків, 10 джерел.

Дипломний проект присвячений розробці інформаційної системи для трансляції Nib-файлів (Xib, Storyboard) у декларативний SwiftUI код. Метою створення системи є спрощення процесу переводу користувацького інтерфейсу від Nib-ів до SwiftUI.

У розділі загальних положень описано предметне середовище, приводяться основні процеси, наведено варіанти використання програмного продукту та функціональні вимоги до нього. Визначено мету розробки та поставлено необхідні для її вирішення задачі.

У розділі інформаційного забезпечення визначено вхідні та вихідні дані для програмного продукту, описані відмінності режимів роботи для різних форматів вхідних файлів та різної їх структури.

Програмне забезпечення представляє опис засобів розробки програмного забезпечення та визначає його архітектуру. Описано схему генерації вихідного коду з даних структур даних.

У технологічному розділі описано виконання перевірки програмного забезпечення відповідності функціональним вимогам. Описано порядок виконання тестів.

КЛЮЧОВІ СЛОВА: SWIFTUI, NIB, ТРАНСЛЯТОР, UI, APPLE, XIB, STORYBOARD

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of four sections, contains 14 tables, 3 appendices, 10 sources.

The diploma project is devoted to the development of an information system for translating Nib-files (Xib, Storyboard) into declarative SwiftUI source code. The purpose of creating the system is to simplify the process of translating the user interface from Nibs to SwiftUI.

The General Terms section describes the subject matter, lists the basic processes, provides main use-cases for the product and functional requirements to it. The purpose of development is defined and the necessary tasks for solving it are set.

In the information support section, the input and output data for the software product is defined, the differences of operation modes for various formats of input files and their different structure are described.

Software provides a description of the software development tools and defines its architecture. The scheme of source code generation from input data structures is described.

The technology section describes the process of software verification for compliance with functional requirements. The order of the tests execution is described.

KEY WORDS: SWIFTUI, NIB, TRANSLATOR, UI, APPLE, XIB, STORYBOARD

ВМЕСТО ЭТОГО ЛИСТА ВСТАВИТЬ ЛИСТ ТИТУЛА ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

– англ. – англійською;

– swift – Багатопарадигмова мова загального призначення, розроблена Apple для написання додатків під платформи macOS, iOS та інших систем;

– xml – мова розмітки, що визначає набір правил для представлення документів у форматі, що зрозумілий як комп’ютеру, так і людині;

– дерево (деревовидна структура даних) – спосіб представлення ієрархічної за натурою структури у графічній формі;

– парсинг – процес аналізу рядків символів з людської мови, комп’ютерної мови чи структур даних, що співвідносяться з правилами формальної граматики;

– ide (Інтегроване середовище розробки) - це програмне забезпечення, яке надає комплексні засоби для розробки програмного забезпечення комп'ютерним програмістам. IDE зазвичай складається щонайменше з редактора вихідного коду, засобів автоматизації збирання проекту та дебагера;

– swiftui – це інструментарій для розробки UI для Apple додатків із використанням декларативного підходу;

– nib файл - це особливий тип файлу ресурсів, який використовується для зберігання користувальницьких інтерфейсів додатків iOS та Mac. Файл nib - це документ Interface Builder;

– mvc (Model-View-Controller) шаблон дизайну призначає об’єктам у програмі одну з трьох ролей: model, view або controller. Шаблон визначає не тільки ролі що об'єкти грають у додатку, він визначає спосіб спілкування об'єктів один з одним. Кожен з трьох типів об'єктів відокремлений від інших абстрактними межами та спілкується з об'єктами інших типів через ці межі. Колекцію об'єктів певного типу MVC у додатку іноді називають шаром - наприклад, шаром моделі.;

– autolayout - система компонування View на основі обмежень. Це дозволяє розробникам створювати адаптивний інтерфейс, який відповідним чином реагує на зміни розміру екрана та орієнтації пристрою;

– codable – Протокол для структур що дозволяє реалізовувати серіалізацію XML та JSON;

ВСТУП

Із розвитком інформаційних систем обчислювальні пристрої різних типів набувають усе більшого розповсюдження: комп’ютери, планшети, телефони, годинники, телевізори та інші. Вони дозволяють користувачам вирішувати все більше і більше робочих та креативних задач: від набору текстів до малювання і створення відео. Так само зростає і кількість доступного програмного забезпечення для усіх видів пристроїв, зявляються магазини додатків такі як Apple App Store, Google Play, Windows Store. Розробка і продаж програмного забепечення стає новим прибутковим ринком і станом на кінець 2019 року у найбільш популярних магазинах було опубліковано близько 2 мільйонів програм. Розробкою додатків займаються вже не тільки великі компанії але й окремі розробники, невеликі команди. Із новими засобами розробки, багатофункціональними зручними IDE та великою кількістю потужних фреймворків розробка програмного забезпечення стає досить швидким та динамічним процесом. Користувачі очікують щотижневих оновлень додатків із виправленнями та новими функціями. Постійна зміна умов діяльності змушує розробників пристосовуватися адекватно реагувати до нових викликів в умовах сильної конкуренції.

Більша кількість додатків на сьогоднішній день працюють у графічному режимі. Вони прийшли на заміну консольним додаткам що були колись розповсюджені і витіснили їх. Звісно консольні застосунки також використовуються проте набагато рідше ніж раніше і, зазвичай, лише в обмеженому колі просунутих користувачів. Такі програми відійшли на задній план поступившись місцем своїм більш зручним аналогам. Користувачі вже звикли до інформативних, красивих інтерфейсів. Часто якісний і зручний UI стає вирішальним фактором при виборі того чи іншого застосунку. Тож задача створення UI для додатків є досить значущою і неймовірно важливою частиною процесу розробки якою не можна нехтувати.

Більшість IDE мають спеціальні “констуктори інтерфейсу” що дозволяють зручно розташувати елементи керування (кнопки, списки, перемикачі та інше) на екрані. Не є вийнятком є і Xcode (у минулому Project Builder), що використовується для побудови програми під системи Apple:

– macOS (Mac OS X у минулому) - для компютерів;

– iOS - для телефонів і планшетів;

– tvOS - для телевізорів;

– watchOS - для смарт годинників.

Для побудови UI під усі ці системи Xcode має спеціальний редактор - Interface Builder. Він був створений у NeXT більше 30 років тому і використовується по сьогоднішній день. Interface Builder працює із Nib файлами: .xib та .storyboard. Nib-и - це великі текстові XML файли що представляють собою складну деревовидну структуру View із параметрами як от: кольори, розміри елементів, стилі і т п. Xib та Storyboard файли відрізняються лише тим що Storyboard-и мають декілька сцен (екранів) а Xib файл представляє собою лише один екран (або View). Використання такого підходу набуло популярності свого часу у порівнянні зі створенням UI у рантаймі за допомогою коду через простоту і наглядність. Проте є і недоліки. Так, при паралельній роботі командою, виникає багато конфліктів у Nib-файлах. Зазвичай виправляти їх треба вручну і це - досить складна задача. Також ефективно працювати із файлами такого типу практично не можливо без Xcode.

Існують декілька альтернативних варіантів:

– реалізація UI із коду - можна збирати ієрархію UI елементів у рантаймі пишучи в імперативному стилі. Так вдасться уникнути вищезазначених проблем із Nib-ами, проте отримаємо недоліки як от: неможливість одразу бачити структуру UI, складнощі із NSLayoutConstrains;

– flutter / React Native / Xamarin та подібні альтернативні рішення - потребують повного переписування проекту на іншу мову і фреймворк та не гарантують довготривалу підтримку та наявність усіх потрібних API. Їх рідко використовують і в цілому такі підходи принесуть більше проблем ніж покращень.

Втім, дійсно гарна альтернатива для нативних iOS/macOS/watchOS/tvOS проектів зявилася лише у 2019 році із релізом SwiftUI - DSL на основі багатопарадигменної мови Swift для побудови декларативного UI. SwiftUI вирішує ряд проблем Nib-ів і при цьому не має проблем вищезазначених підходів. Тож SwiftUI в цілому був позитивно сприйнятий розробниками. Зараз багато нових додатків уже починають писати з використанням SwiftUI. Проте аби переписати старі проекти на SwiftUI необхідно витратити багато часу на роботу із UI. Це займає не мало часу і є складною задачею, особливо коли технологія є новою для розробників. Адже треба створити аналогічний UI з використанням нового подходу і при цьому зберегти усі стилі, кольори і розміри. Більш того ряд UI елементів що були доступні при роботі із Nib-ами, не мають прямих аналогів у SwiftUI - їх треба реалізовувати окремо. Ну і звісно у цього підходу є ряд обмежеть та нюансів. Усе це додає процесу складності і може будти вирішальним фактором у бік відмови від оновлення проекту і продовження роботи за старою схемою.

Видно що є необхідність автоматизації і спрощення задачі переходу від Nib-ів до SwiftUI. Для цього потрібне зручне програмне забезпечення що прийматиме на вхід старі Nib-файли а на виході видаватиме SwiftUI View файли зі структурою і стилями що відповідають початковим. Звісно це не автоматизує процесс повністю, проте допоможе із великим обємом однотипної роботи по конвертації старого коду та відтворенню уже існуючих структур. Це тим самим сильно зекономить час для розробників.

Практичне значення одержаних результатів

Розроблено програмне забезпечення що транслює Nib-файли у SwiftUI декларативний UI код, при цьому зберігає структуру і стилі елементів, автоматично додає шаблони байндінгів для тих елементів що їх потребують і шаблони коду для обробки вводу користувача, додає обгортки для UIKit / AppKit елементів що все ще відсутні у SwiftUI. Цим самим сильно спрощується процес переходу до нового підходу на існуючих проектах.

Публікації

Віхляєв О. О. Трансляція UIKit, AppKit користувацького інтерфейсу із XIB у SwiftUI // Всеукраїнська науково-практична конференція молодих вчених та студентів “Інформаційні системи та технології управління – ІСТУ-2020”.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Було проаналізовано процес переводу проектів з Nib-файлів на SwiftUI код. В ході аналізу було виявлено що багато часу займає:

- відтворення існуючих сцен та View зі збереженням властивостей;
- написання обгорток для елементів що не доступні у SwiftUI.

Також видно що така робота по оновленню проекту є досить однотипною і може бути автоматизована.

1.2 Змістовний опис і аналіз предметної області

Конвертер із Nib файлів у SwiftUI код є досить простим у користуванні Cocoa-додатком із єдиним вікном. Користувач обирає вхідний файл - на вибір або Xib або Storyboard. Можливий вибір і декількох файлів. Після цього відбувається конвертація усіх файлів і користувачу надається можливість вибору директорії для експорту результатів. Кожна сцена або View експортується як окремий .swift файл і усі загально необхідні компоненти експортуються у одну папку.

1.3 Аналіз успішних IT-проектів

1.3.1 Аналіз відомих технічних рішень

У ході пошуку і аналізу рішень для кодогенерації SwiftUI коду виявилося що на даний момент рішень немає, тож потрібно реалізовувати таку функціональність з нуля.

Для трансляції Nib-ів у SwiftUI код необхідно спершу серіалізувати XML розмітку вхідного Xib / Storyboard файлу. Для цього існує спеціальна бібліотека IBDecodable, написана мовою Swift і доступна для підключення через Swift Package Manager. Код бібліотеки відкритий і передбачена можливість дописати необхідну функціональність чи виправити помилки.

1.3.2 Аналіз відомих програмних продуктів

SWIFTIFY (Storyboard => SwiftUI) - єдиний на даний момент доступний конвертер. Він реалізований як веб-сайт, нативних додатків не має. Його можна знайти за посиланням: Swiftify. З опису видно що конвертер обмежений файлами розміром до 1 КБ. Якщо ж зареєструватися, то ліміт буде збільшено до 2 КБ. Однак пустий Storyboard файл має розмір близько 2 КБ, отже використовувати конвертер без реєстрації фактично неможливо. Після проходження реєстрації і завантаження до Swiftify декількох тестових файлів, виявилося що навіть із максимально простими Nib-файлами допустимого розміру конвертер не працює взагалі. На виході маємо пусту SwiftUI структуру що має всередині лише Spacer() - елемент для генерації відступів і коментар наступного змісту: "// TODO: replace with the actual content". Тож на момент аналізу конвертер фактично існує, проте не функціонує жодним чином.

Втім, за умови функціонування цього аналога, можна було б віднести до його переваг те що він доступний як веб-додаток із більшості сучасних браузерів і не вимагає встановлення будь-якого додаткового програмного забезпечення. Але з іншого боку при відсутності інтернет-підключення це стає його великим недоліком адже користуватися застосунком буде неможливо. Більш того переваги веб-додатків як от доступність на усіх системах що мають доступ до веб-ресурсу нівелюються тим що задачі роботи з Nib файлами виникають переважно при роботі із додатками для Apple-систем робота над якими ведеться виключно на macOS.

1.4 Аналіз вимог до програмного забезпечення

Процес роботи з додатком буде складатися з таких етапів:

– вибір вхідних файлів;

– серіалізація вхідних файлів;

– аналіз і конвертація у Swift файли;

– експорт вихідних файлів.

Актором системи є лише Клієнт. У таблиці 1.1 наведені його дії при роботі із програмою.

Таблиця 1.1 – Актори системи

Актор	Варіант використання	Опис дії варіанта використання
Клієнт	Імпорт Nib файлів	1. Користувач натискає кнопку “Обрати Nib файл” 2. Програма відкриває вікно файлового менеджера і пропонує обрати файли 3. Користувач переходить у необхідну директорію і обирає .xib і/або .storyboard файли, натискає кнопку підтвердження вибору 4. Програма зчитує вхідні файли
Клієнт	Налаштування необхідних для конвертації файлів, сцен і View	
Клієнт	Експорт результату	

На рисунку 1.1 наведено діаграму варіантів використання.

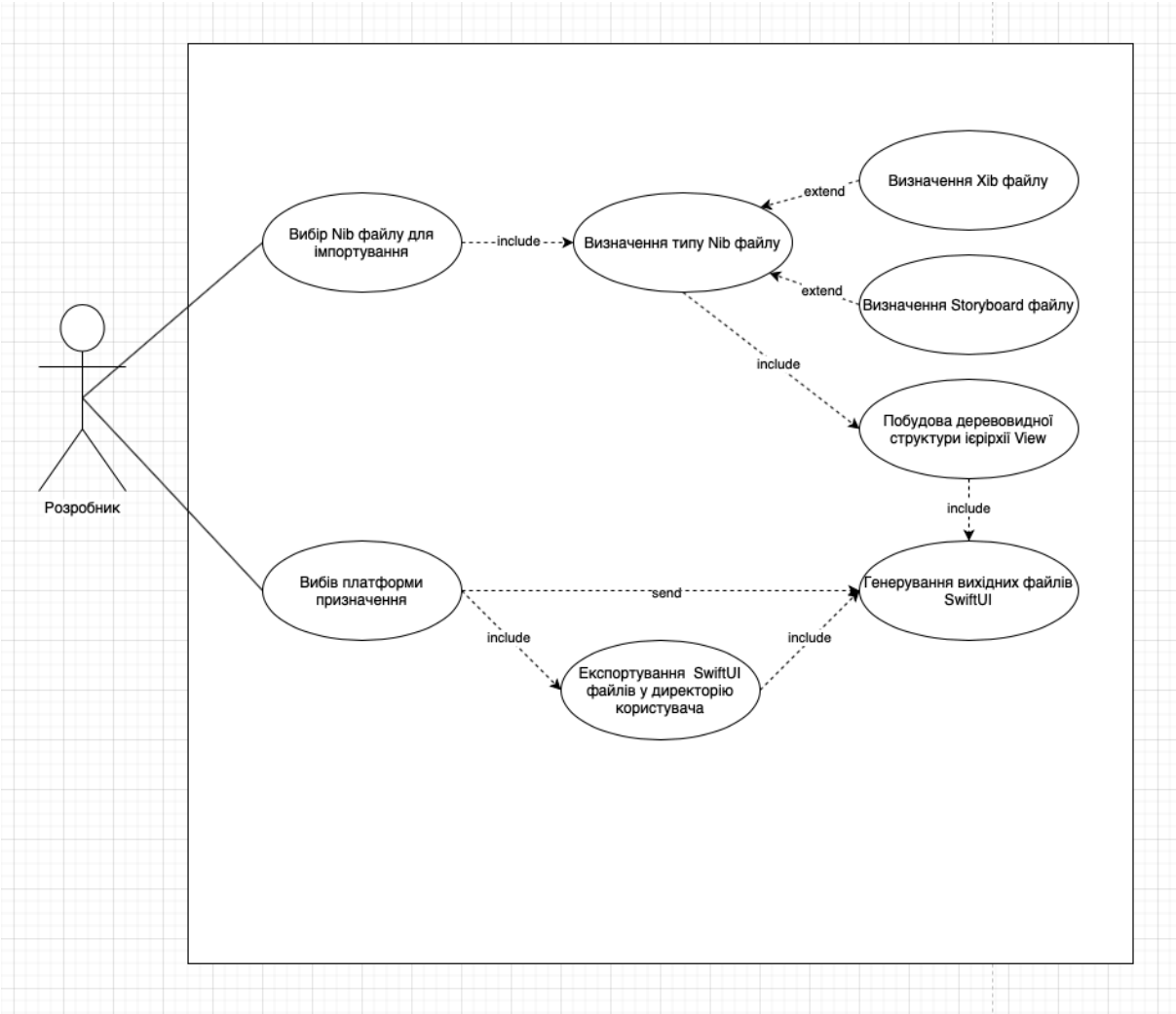
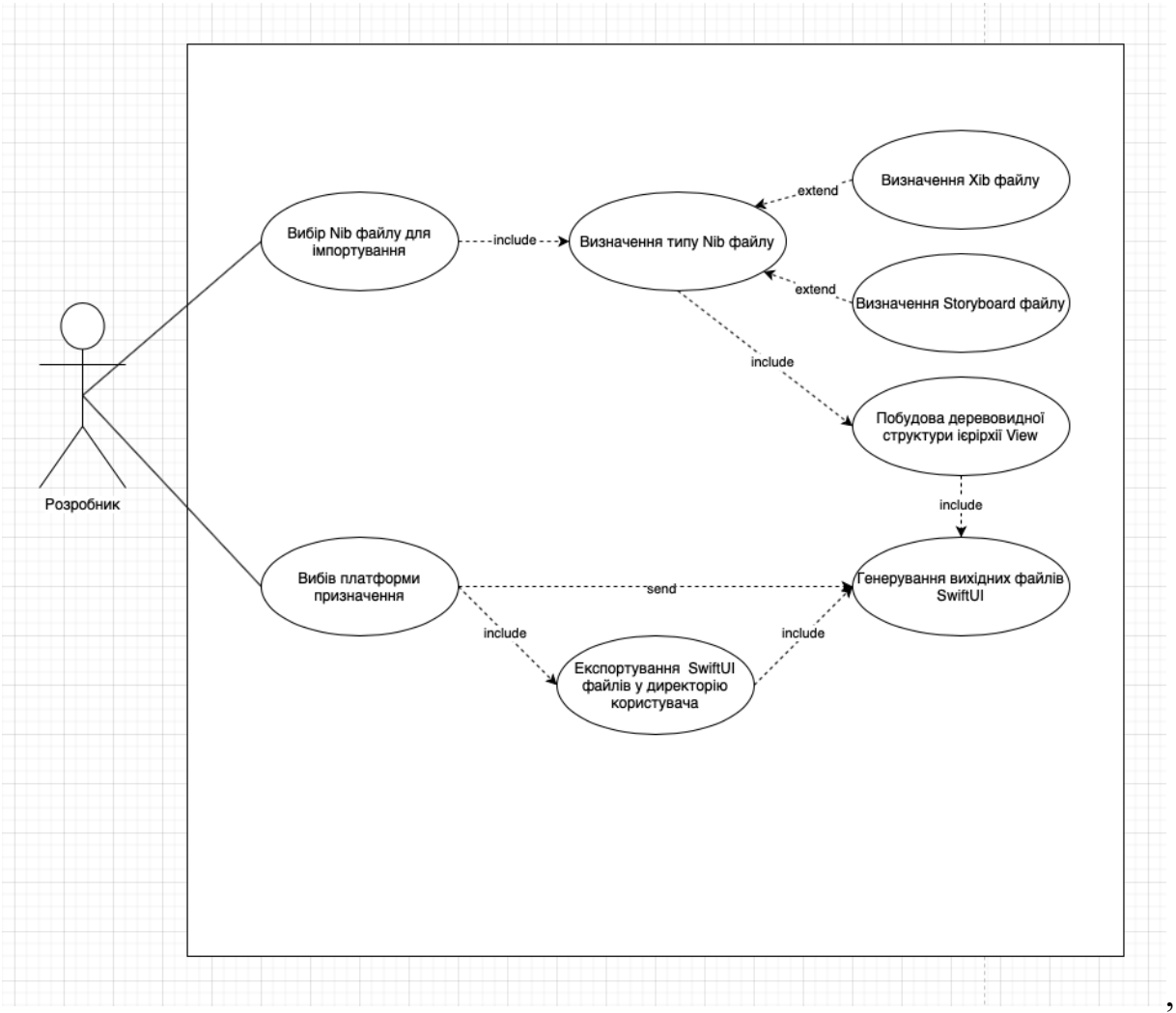


Рисунок 1.1 – Схема структурна варіантів використання

1.5 Аналіз вимог до програмного забезпечення

Програмне забезпечення надає користувачу можливість імпортувати Nib-файли та експортувати результати трансляції.

Ілюстрацію процесу використання можна побачити нижче на



, де для зображення опцій була використана Діаграма варіантів використання.

1.5.1 Розроблення функціональних вимог

1.5.1.1 Тип процесору

Intel x86-64, сумісний із macOS Catalina

1.5.1.2 Об’єм ОЗП

Кількість оперативної пам’яті достатня для нормальної роботи ОС (не менше 4 ГБ).

1.5.1.3 Об’єм жорсткого диску

Вільне місце на жорсткому диску (зазвичай не менше 20 МБ)

1.5.1.4 Операційна система

Операційна система macOS 10.15 Catalina і вище

1.5.1.5 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

- імпортування файлів із допомогою Finder NSOpenPanel;
- налаштування для вибору платформи призначення;
- експортування результату із допомогою Finder NSOpenPanel.

1.5.2 Розроблення нефункціональних вимог

Розроблене програмне забезпечення є macOS графічним додатком що автоматизує процес переводу проектів від Nib до SwiftUI. Дане програмне забезпечення побудоване за MVC парадигмою у об’єктно-орієнтованому стилі. Додаток можна запустити на macOS 10.15 Catalina і вище.

Мова програмного забезпечення: Англійська

Програма повинна мати максимально простий і зрозумілий користувацький інтерфейс у стандартному для операційної системи стилі для забезпечення високої продуктивності роботи.

1.5.3 Постановка комплексу завдань модулю

Метою розроблення даного програмного продукту є автоматизація процесу переводу проектів від Nib до SwiftUI. Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

1.5.3.1.1 Спрощення і пришвидшення процесу переходу:

– від Storyboard сцен до SwiftUI екранів;
– від Xib View до SwiftUI View.

1.5.3.1.2 Зменшення стилістичних помилок при переході до нового UI;

1.5.3.2 Для реалізації поставлених цілей програмне забезпечення має вирішувати наступні задачі:

– трансляція налаштувань UI елементів (стани, режими роботи);
– трансляція стилів (кольорів, шрифтів);
– трансляція відступів, розмірів та NSLayoutConstraints (якщо це не суперечить концепціям SwiftUI);
– трансляція UIKit / AppKit елементів що не мають SwiftUI аналогів;
– автоматичне вирішення основних проблем SwiftUI (наприклад ліміт на 10 вкладених View);
– генерація добре форматованого коду;
– автоматична генерація @State змінних для елементів що потребують байндінги.

1.6 Висновки по розділу

Отже, у даному розділі був приведений опис предметного середовища розробки. Описано процеси діяльності, в яких визначено головні бізнес-процеси, що будуть виконані системою. Описано функціональну модель системи, приведено приклади основних варіантів використання додатку. Визначено призначення, мету та задачі що стоять перед розробкою.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Бізнес процес можна для даної утиліти можна описати наступним чином. Єдиною дією, що доступна користувачу є імпорт Nib-файлу. Для здійснення цього необхідно натиснути відповідну кнопку і у списку файлів обрати необхідний. Для вибору доступні лише файли із розширеннями .xib та .storyboard.

Наступним кроком є серіалізація вхідного файлу і трансляція усіх сцен і/або View у ньому. Результуюча структура зберігається у пам’яті, користувач отримує можливість експортувати результат.

Для описання процесів функціонування програмного забезпечення було використано максимально зручну нотацію опису бізнес-процесів, а саме – BPMN. Структурна схема бизнес-процессу трансляції файлів наведена на рисунку 2.1.

Спочатку користувач запускає програму і обирає Nib-файл для імпортування. Далі відбувається зчитування файлу із диску і у разі помилки – відображається відповідне повідомлення, а додаток переходить у початковий стан. Якщо ж зчитування відбулося успішно – визначається тип файлу (Xib чи Storyboard). Якщо маємо Storyboard файл – серіалізуємо усі сцени із усіма View. Якщо маємо Xib файл то серіалізуємо єдину кореневу View. Будуємо деревовидну структуру ієрархії View.

Далі у користувача є можливість обрати платформи для яких потрібно генерувати вихідний код. За замовчуванням – iOS. Але можна обрати ще і macOS.

Наступним кроком є експорт результату. Для цього користувач натискає кнопку “Export” та обирає директорію призначення. Після цього генерується вихідний код і зберігається на диск.

Рисунок 2.1- Схема структурна діяльності

2.2 Архітектура програмного забезпечення

Програмне забезпечення спроектовано і побудовано у ООП стилі згідно з Apple MVC парадигмою. MVC є основою для гарного дизайну Cocoa застосунків і добре інтегрується із усіма фреймворками Apple. Схема наведена на Рисунку 2.2:

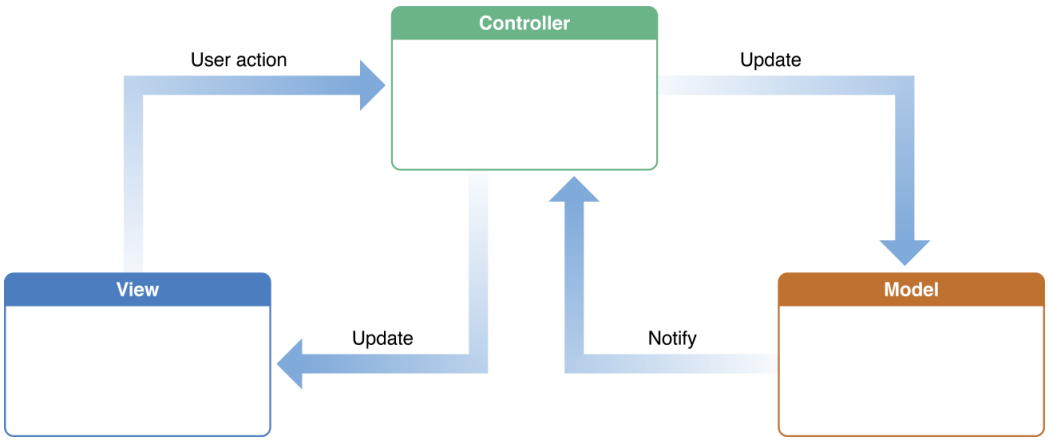


Рисунок 2.2 – Apple MVC

Головним класом що відповідає за трансляцію є AVNibReader. Він має методи для зчитування файлів із диску, серіалізації XML. Також він має методи для трансляції серіалізованих View у наступні структури:

- avview;
- avlabel;
- avbutton;
- avtextfield;
- avswitch;
- avslider;
- avactivityindicatorview;
- avstepper;
- avprogressview;

- avtableview;
- avtableviewcellcontentview;
- avimageview;
- avtabbar;
- avtoolbar;
- avsegmentedcontrol;
- avstackview.

Тож визначивши назву класу елементу і отримавши його параметри із XML, можна передати параметри у структури зазначені вище і отримати готові для об’єднання у дерево елементи.

Завдяки ООП підходу набір підтримуваних View можна з легкістю розширювати шляхом додавання структур даних для нових елементів і включення перевірки на відповідність імені нового класу у методі *func translateView(from view: ViewProtocol)*.

На випадок виникнення помилок в процесі роботи AVNibReader має enum AVNibReaderError із наступними case-ами:

- noRootView;
- noScenes;
- cantGenerateSwiftUIData;
- unsupportedFileType.

Він реалізує протокол Error і змінну *localizedDescription* що залежно від типу помилки повертає рядок що описує помилку.

Базовим класом для усіх елементів є AVBaseView. Оскільки UI – це дерево елементів то у базовому класі визначено змінну *subviews* - масив View вкладених у дану. Також у AVBaseView є computed змінна *stateBindings* що за замовчуванням повертає пустий масив і може бути перевизначена у нащадках таким чином аби повертати потрібні для View байндінги. Поряд маємо final computed змінну *allSatateBindings* що повертає усі байндінги для даної ієрархії. Аналогічно із змінними *customClasses* та *allCustomClasses*. Вони повертають

масиви типу [AVCustomClass]. Елементи масиву – блоки коду призначені для однієї чи декількох платформ (визначається за домомогою enum OSTarget). Потреба у виокремлені таких блоків коду з'являється через те що деякі відсутні у SwiftUI елементи реалізовані у вигляді окремих структур під різні платформи (iOS / macOS).

Структура класів наведена на Рисунку 2.3:

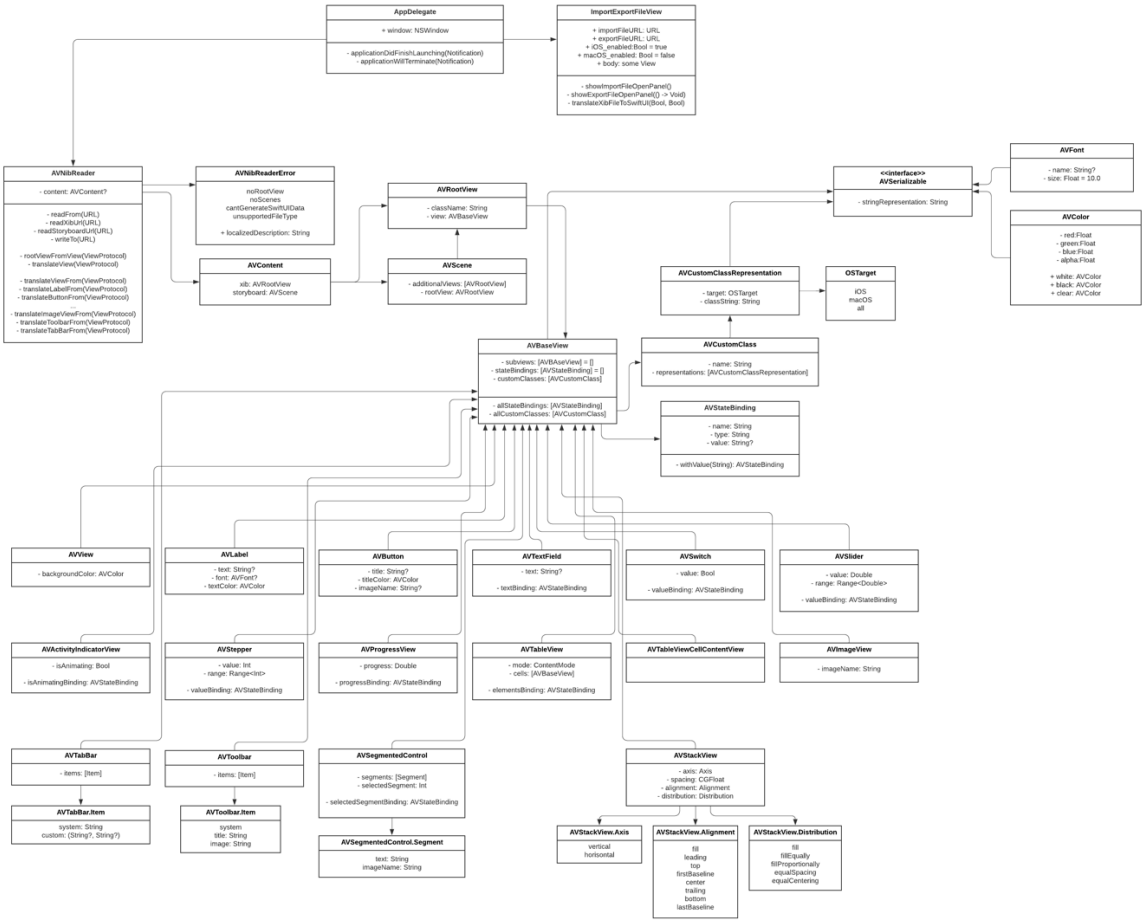


Рисунок 2.3– Схема структурна класів програмного забезпечення

2.2.1 Вхідні дані

На вхід приймаються Nib файли двох типів: Xib та Storyboard. Обидва являють собою складної структури XML розмітку. Далі наведено приклади пустого Xib файлу та Storyboard файлу із двома сценами.

Xib:

```
<?xml version="1.0" encoding="UTF-8"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="3.0"
toolsVersion="15705" targetRuntime="iOS.CocoaTouch" propertyAccessControl="none"
useAutolayout="YES" useTraitCollections="YES" useSafeAreas="YES" colorMatched="YES">
   <device id="retina6_1" orientation="portrait" appearance="light"/>
   <dependencies>
      <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="15706"/>
      <capability name="Safe area layout guides" minToolsVersion="9.0"/>
      <capability name="documents saved in the Xcode 8 format" minToolsVersion="8.0"/>
   </dependencies>
   <objects>
      <placeholder placeholderIdentifier="IBFilesOwner" id="-1" userLabel="File's Owner"
customClass="EmptyVC" customModule="TestXibProject" customModuleProvider="target">
         <connections>
            <outlet property="view" destination="i5M-Pr-FkT" id="sfx-zR-JGt"/>
         </connections>
      </placeholder>
      <placeholder placeholderIdentifier="IBFirstResponder" id="-2"
customClass="UIResponder"/>
      <view clearsContextBeforeDrawing="NO" contentMode="scaleToFill" id="i5M-Pr-FkT">
         <rect key="frame" x="0.0" y="0.0" width="414" height="896"/>
         <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
         <viewLayoutGuide key="safeArea" id="fnl-2z-Ty3"/>
         <point key="canvasLocation" x="137.68115942028987" y="137.94642857142856"/>
      </view>
   </objects>
</document>
```

Storyboard:

```
<?xml version="1.0" encoding="UTF-8"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.XIB" version="3.0"
toolsVersion="15705" targetRuntime="iOS.CocoaTouch" propertyAccessControl="none"
useAutolayout="YES" useTraitCollections="YES" useSafeAreas="YES" colorMatched="YES">
   <device id="retina6_1" orientation="portrait" appearance="light"/>
```

```
    <dependencies>
        <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="15706"/>
        <capability name="Safe area layout guides" minToolsVersion="9.0"/>
        <capability name="documents saved in the Xcode 8 format" minToolsVersion="8.0"/>
    </dependencies>
    <scenes>
        <!--View Controller-->
        <scene sceneID="H9G-sy-Y6c">
            <objects>
                <viewController id="z0v-cL-QdK" sceneMemberID="viewController">
                    <view key="view" contentMode="scaleToFill" id="4kW-bi-04A">
                        <rect key="frame" x="0.0" y="0.0" width="414" height="896"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES"
heightSizable="YES"/>
                        <color key="backgroundColor" systemColor="systemBackgroundColor"
cocoaTouchSystemColor="whiteColor"/>
                        <viewLayoutGuide key="safeArea" id="38u-0e-nwX"/>
                    </view>
                </viewController>
                <placeholder placeholderIdentifier="IBFirstResponder" id="Nnx-WN-uJr"
userLabel="First Responder" customClass="UIResponder" sceneMemberID="firstResponder"/>
            </objects>
            <point key="canvasLocation" x="-103" y="155"/>
        </scene>
        <!--View Controller-->
        <scene sceneID="A3g-qQ-QTf">
            <objects>
                <viewController id="gKL-V3-aBm" sceneMemberID="viewController">
                    <view key="view" contentMode="scaleToFill" id="Bvp-BW-OYP">
                        <rect key="frame" x="0.0" y="0.0" width="414" height="896"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES"
heightSizable="YES"/>
                        <color key="backgroundColor" systemColor="systemBackgroundColor"
cocoaTouchSystemColor="whiteColor"/>
```

```
                <viewLayoutGuide key="safeArea" id="OeM-E6-ReZ"/>
            </view>
        </viewController>
        <placeholder placeholderIdentifier="IBFirstResponder" id="nTv-X7-xXf"
userLabel="First Responder" customClass="UIResponder" sceneMemberID="firstResponder"/>
    </objects>
    <point key="canvasLocation" x="624" y="155"/>
  </scene>
 </scenes>
</document>
```

2.2.2 Вихідні дані

Вихідними даними є набір SwiftUI файлів: файли для Xib View / Storyboard сцен що були імпортовані із preview структурами та файли структур для забезпечення сумісності елементів що не мають SwiftUI аналогів.

Для одного пустого Xib файла матимемо вихідний .swift файл із наступним вмістом:

```
import SwiftUI
struct UIView: View {
    var body: some View {
        ZStack {
            Color.clear.edgesIgnoringSafeArea(.all)
        }
    }
}

struct UIView_Previews: PreviewProvider {
    static var previews: some View {
        UIView()
    }
}
```

2.3 Опис елементів і структур даних для роботи із ними

2.3.1 Опис UIView елементів

View - це основна складова користувацького інтерфейсу iOS програми, а клас UIView визначає поведінку, спільну для всіх View. Об'єкт View відображає певний вміст у межах свого прямокутника-границі та обробляє будь-які взаємодії з цим вмістом. Вміст може бути як намалованим за домомогою CoreGraphics так і відображеним іншими View що вкладені у дану. Для відображення міток, зображень, кнопок та інших елементів інтерфейсу, які зазвичай зустрічаються в додатках, використовуються підкласи, передбачені UIKit, що описані у наступних підрозділах.

Кожна сцена та коренева View із Nib файлів буде трансльована у SwiftUI структуру що реалізує протокол SwiftUI.View. Що до усіх інших View то SwiftUI не має прямого їх аналогу для використання у UI ієрархіях, проте найближчим за функціями елементом є ZStack - стек в якому декілька елементів можуть розташовуватись у декілька шарів. Для представлення складних обмежень і встановлення залежностей розмірів можна використовувати вкладену структуру GeometryReader. Інші ж View можна заміняти на простіші у роботі VStack / HStack із використанням Spacer структур і відступів на вкладених елементах.

2.3.2 Опис UILabel елементів

UILabel - це класс View що відображає текст. Параметри відображення налаштовуються змінними екземпляру классу. У SwiftUI є аналог до UILabel - структура Text. Як параметр вона приймає текст а налаштовується за допомогою модифікуючих функцій. У таблиці 2.1 наведено основні відповідності між UILabel та Text:

Таблиця 2.1 – Відповідність між UILabel та Label

UILabel	**Опис**	**Text**	**Опис**
text	текст що буде відображений	init(verbatim: String)	
attributedText	аналогічно до text, але приймаються NSAttributedString	—	Реалізуються за допомогою модифікаторів і конкатенації
font	шрифт тексту	func font(Font?)	шрифт тексту
textColor	колір тексту	func foregroundColor(Color?)	колір тексту
textAlignment	вирівнювання тексту	func multilineTextAlignment(TextAlignment)	вирівнювання тексту
lineBreakMode	визначає яким чином будуть оброблені переноси рядків	func truncationMode(TruncationMode)	визначає яким чином будуть оброблені переноси рядків
numberOfLines	максимальна кількість рядків	func lineLimit(Int?)	максимальна кількість рядків

2.3.3 Опис UIButton елементів

UIButton - елемент що виконує певну дію у відповідь на взаємодію користувача (натискання / відпускання / інше). Параметри відображення налаштовуються змінними екземпляру классу. У SwiftUI є аналог до UIButton - структура Button. У таблиці 2.2 наведено відповідності між ними:

Таблиця 2.2 - Відповідність між UIButton та Button

UIButton	**Опис**	**Button**	**Опис**
func title(for: UIControl.State) -> String? func setTitle(String?, for: UIControl.State)	Повертає / встановлює текст кнопки	—	Реалізується конфігурацією контенту кнопки
func titleColor(for: UIControl.State) -> UIColor? func setTitleColor(UIColor?, for: UIControl.State)	Повертає / встановлює колір тексту кнопки	—	
func image(for: UIControl.State) -> UIImage? func setImage(UIImage?, for: UIControl.State)	Повертає / встановлює картинку кнопки	—	

Навідміну від Text що конфігурується модифікуючими функціями у Button зовнішній вигляд цілком залежить від вигляду контенту. Структура створюється таким конструктором: *init(action: () -> Void, label: () -> Label)* перший параметр якого - це замикання що буде викликано при спрацьовуванні кнопки а другий - контент кнопки. Наприклад для простої текстової кнопки може бути передано

лише Text структуру а для кнопки із текстом і картинкою відповідно Text і Image структури. Такий підхід додає гнучкості адже таким чином можна сконструювати кнопку із довільної кількості картинок і заголовків. У UIKit це було набагато складніше.

2.3.4 Опис UISegmentedControl елементів

UISegmentedControl - елемент що складається із декількох секцій-кнопок, відображає виділеною одну із них і при зміні відділення сповіщає про це. Кожен із сегментів може мати в собі текст або зображення. У SwiftUI аналогічний елемент - Picker. Основні їх відмінності зазначено у таблиці 2.3:

Таблиця 2.3 - Відповідність між UISegmentedControl та Picker

UISegmentedControl	**Опис**	**Picker**	**Опис**
selectedSegmentIndex	Індекс виділенного сегмента	—	Передається у конструкторі

У UISegmentedControl виділення задається присвоєнням нового значення змінній екземпляру а у SwiftUI Picker виділення залежить від стану змінної-байндінгу що передається у конструкторі: *init<S>(S, selection: Binding<SelectionValue>, content: () -> Content).* Про зміну виділення у UIKit елемента можна дізнатися за патерном Target-Action, а у SwiftUI - за зміною Binding змінної.

2.3.5 Опис UITextField елементів

UITextField - текстові поля що використовуються для збору тексту на основі вводу за допомогою екранної клавіатури. Клавіатуру можна налаштувати для багатьох типів введення, таких як звичайний текст, електронні листи, номери

тощо. Його SwiftUI аналог - TextField. У таблиці 2.4 наведено основні відповідності між ними:

Таблиця 2.4 - Відповідність між UITextField та TextField

UITextFiel d	**Опис**	**TextField**	**Опис**
text	Визначає початковий текст	—	Байндінг змінна передється у конструкторі
placeholder	Визначає підказку що відображається при відсутності тексту	—	Передється у конструкторі
font	Шрифт тексту	func font(Font)	Шрифт тексту
textColor	Колір тексту	func foregroundColor(Color)	Колір тексту
textAlignme nt	Вирівнювання тексту	func multilineTextAli gnment(TextAlig nment)	Вирівнювання тексту
borderStyle	Стиль рамки поля	func textFieldStyle(Te xtFieldStyle)	Стиль рамки поля
textContent Type	Тип вмісту	func textContentType( )	Тип вмісту

Текстові поля UIKit використовують механізм Target-Action та об’єкт делегування для повідомлення про зміни, внесені під час редагування та для керування додатковою функціональністю. SwiftUI TextField же сповіщає про зміну стану викликом замиканнь що передаються у конструкторі: *init<S>(S, text:Binding<String>, onEditingChanged: (Bool) -> Void, onCommit: () -> Void)* і налаштовується викликом модифікуючих функцій.

2.3.6 Опис UISlider елементів

UISlider - елемент для вибору одного значення на неперервному проміжку значень. Вибір виконується пересуванням слайдеру. Конфігурується мінімальним і максимальним значенням проміжку, поточним значенням, зображенням для максимального і для мінімального значення. Має SwiftUI аналог - Slider. Основні відповідності між елементами наведено у таблиці 2.5:

Таблиця 2.5 - Відповідність між UISlider та Slider

UISlider	**Опис**	**Slider**	**Опис**
value	Поточне значення слайдеру	—	Визначене значенням змінної байндінгу
minimumValue	Нижня границя діапазону	—	Діапазон передається у конструкторі як ClosedRange<V>
maximumValue	Верхня границя діапазону	—	
minimumValueImage	Зображення для нижньої границі	—	Заміняється вкладенням структур Image та Slider у HStack
maximumValueImage	Зображення для верхньої границі	—	

У UIKit зміни відстежуються за патерном Target-Action а у SwiftUI викликаються замикання що передаються у конструкторі структури: *init<V>(value: Binding<V>, in: ClosedRange<V>, onEditingChanged: (Bool) -> Void)*.

2.3.7 Опис UIActivityIndicatorView елементів

UIActivityIndicatorView - View для відображення прогрессу виконання операції. Можна запустити і зупинити анімацію. Має декілька стилів індикатора і налаштування кольору. SwiftUI не має аналогу для такого елементу, тож необхідно обернути UIKit елемент у нову структуру що реалізує протокол UIViewRepresentable для iOS та NSViewRepresentable для macOS. Отримали такі структури:

```
#if os(iOS)

struct ActivityIndicator: UIViewRepresentable {

   @Binding var isAnimating: Bool

   func makeUIView(context: UIViewRepresentableContext<ActivityIndicator>) ->
UIActivityIndicatorView {
      return UIActivityIndicatorView(style: .medium)
   }

   func updateUIView(_ uiView: UIActivityIndicatorView, context:
UIViewRepresentableContext<ActivityIndicator>) {
      isAnimating ? uiView.startAnimating() : uiView.stopAnimating()
   }

}

#else
```

```
struct ActivityIndicator: NSViewRepresentable {

    @Binding var isAnimating: Bool

    func makeNSView(context: NSViewRepresentableContext<ActivityIndicator>) ->
NSProgressIndicator {
        let indicator = NSProgressIndicator()
        indicator.style = .spinning
        indicator.controlSize = .small
        return indicator
    }

    func updateNSView(_ nsView: NSProgressIndicator, context:
NSViewRepresentableContext<ActivityIndicator>) {
        isAnimating ? nsView.startAnimation(nil) : nsView.stopAnimation(nil)
    }

}

#endif
```

Почати чи зупинити анімацію можна змінивши значення змінної-байндінгу *isAnimating*.

2.3.8 Опис UIProgressView елементів

UIProgressView - ще одна View для відображення прогрессу виконання операції. Може змінювати свій стан прогресу від 0 до 100%. SwiftUI не має аналогу для такого елементу, тож необхідно обернути UIKit елемент у нову структуру що реалізує протокол UIViewRepresentable для iOS та NSViewRepresentable для macOS. Отримали такі структури:

```
#if os(iOS)

struct ProgressView: UIViewRepresentable {
```

```
    @Binding var progress: Double

    func makeUIView(context: UIViewRepresentableContext<ProgressView>) -> UIProgressView {
        let progressView = UIProgressView()
        progressView.progress = Float(progress)
        return progressView
    }

    func updateUIView(_ uiView: UIProgressView, context:
UIViewRepresentableContext<ProgressView>) {
        uiView.progress = Float(progress)
    }

}

#else

struct ProgressView: NSViewRepresentable {

    @Binding var progress: Double

    func makeNSView(context: NSViewRepresentableContext<ProgressView>) ->
NSProgressIndicator {
        let indicator = NSProgressIndicator()
        indicator.style = .bar
        indicator.isIndeterminate = false
        indicator.minValue = 0.0
        indicator.maxValue = 1.0
        indicator.doubleValue = progress
        return indicator
    }
```

```
    func updateNSView(_ nsView: NSProgressIndicator, context:
NSViewRepresentableContext<ProgressView>) {
        nsView.doubleValue = progress
    }
}

#endif
```

2.3.9 Опис UIStepper елементів

UIStepper - елемент що використовується для інкрементування і декрементування значення лічильника. Має SwiftUI аналог - структуру Stepper. Основні відповідності наведено у таблиці 2.6:

Таблиця 2.6 - Відповідність між UIStepper та Stepper

UIStepper	**Опис**	**Stepper**	**Опис**
value	Поточне значення лічильника	—	Визначене значенням змінної-байндінгом
range	Діапазон у якому змінюються значення лічильника	—	Діапазон передається у конструкторі як ClosedRange<V>

Усі параметри передаються у конструкторі: *init<S, V>(S, value: Binding<V>, in: ClosedRange<V>, step: V.Stride, onEditingChanged: (Bool) -> Void)*. При зміні значення лічильника викликається замикання *onEditingChanged.*

2.3.10 Опис UIStackView елементів

UIStackView - View для групування інших View у рядки чи стовпці. Дозволяють використовувати AutoLayout, створюючи інтерфейси користувача, які можуть динамічно адаптуватися до орієнтації пристрою, розміру екрана та до будь-яких змін у доступному для View просторі. UIStackView керує

компонуванням усіх вкладених View. Ці елементи розташовані уздовж осі стека, точний макет залежить від осі, розподілу, вирівнювання, відстані та від інших властивостей. У SwiftUI для цього маємо 2 структури: HStack та VStack для горизонтального та вертикального групування відповідно. Основні відповідності між ними наведено у таблиці 2.7:

Таблиця 2.7 - Відповідність між UIStackView та Stack елементами

UIStackView	**Опис**	**HStack / VStack**	**Опис**
arrangedSubviews	Вкладені View	—	Вирівнюються усі вкладені елементи
axis	Вісь за якою розташовуються елементи	—	HStack - завжди горизонтальний, VStack - вертикальний
alignment	Вирівнювання	init(alignment: Alignment, spacing: CGFloat)	Вирівнювання
spacing	Відступи між елементами		Відступи між елементами

Також у порівнянні із UIStackView HStack та VStack мають наступне обмеження: вони не можуть включати більше ніж 10 вкладених View. Для вирішення цієї проблеми використовуються декілька вкладених Stack-ів в кожному з яких менше за 10 вкладень.

2.3.11 Опис UITableView елементів

UITableView - View для відображення динамічного чи статичного контенту використовуючи різних видів рядки таблиці що скроляться. Таблиці є оптимальним варіантом для відображення великої кількості контенту адже передбачають перевикористання рядків. Для реалізації заповнення контентом і реакцій на користувацькі дії у динамічній таблиці використовується патерн Delegate. У статичних же рядки уже задані в Nib-файлах. У SwiftUI є відповідний

елемент - List. У конструкторі він приймає змінну-байндінг яка зберігає масив даних що потрібно відображати.

2.3.12 Опис UIImageView елементів

UIImageView - View для відображення зображень. Може мати різні режими масштабування контенту. Аналог із SwiftUI - структура Image. У таблиці 2.8 наведено основні відповідності між ними:

Таблиця 2.8 - Відповідність між UIImageView та Image

UIImageView	**Опис**	**Image**	**Опис**
image	Зображення	—	Передається у конструкторі
contentMode	Режим масштабування зображень	func aspectRatio(contentMode: ContentMode)	Режим масштабування зображень

2.3.13 Опис UIToolbar елементів

UIToolbar - елемент інтерфейсу що відображає один і більше елементів керування - UIBarButtonItem. Використовується зазвичай або уздовж верхньої або уздовж нижньої грані екрану. SwiftUI не передбачає альтернатив тож необхідно обернути UIKit елемент у нову структуру що реалізує протокол UIViewRepresentable для iOS та NSViewRepresentable для macOS. Отримали такі структури:

```
class InvocationTarget {

    let action: (() -> Void)?

    init(action: (() -> Void)?) {
        self.action = action
```

```
    }

    @objc func invoke() {
        action?()
    }

}

enum ToolbarItem {
    case system(UIBarButtonItem.SystemItem, (() -> Void)?)
    case title(String, (() -> Void)?)
    case image(String, (() -> Void)?)

    var barButtonItem: UIBarButtonItem {
        switch self {
        case .system(let systemItem, let action):
            let target = InvocationTarget(action: action)
            return UIBarButtonItem(barButtonSystemItem: systemItem, target: target, action:
#selector(InvocationTarget.invoke))
        case .title(let text, let action):
            let target = InvocationTarget(action: action)
            return UIBarButtonItem(title: text, style: .plain, target: target, action:
#selector(InvocationTarget.invoke))
        case .image(let name, let action):
            let target = InvocationTarget(action: action)
            return UIBarButtonItem(image: UIImage(named: name), style: .plain, target: target, action:
#selector(InvocationTarget.invoke))
        }
    }
}

struct ToolbarView: UIViewRepresentable {

    var items: [ToolbarItem]
```

```
    func makeUIView(context: UIViewRepresentableContext<ToolbarView>) -> UIToolbar {
        let toolbar = UIToolbar()
        toolbar.items = items.map({ $0.barButtonItem })
        return toolbar
    }

    func updateUIView(_ uiView: UIToolbar, context:
UIViewRepresentableContext<ToolbarView>) {
        uiView.items = items.map({ $0.barButtonItem })
    }

}
```

2.3.14 Опис UIBarButtonItem елементів

UIBarButtonItem - базовий класс для елементів що відображаються у UIToolbar. Оскільки сам класс UIToolbar для використання зі SwiftUI було обернено у UIViewRepresentable структуру то транслювати самі елементи тулбару немає потреби. Втім, для зручності написання SwiftUI коду було визначено enum ToolbarItem для основних типів елементів.

2.3.15 Опис UITabBar елементів

UITabBar - елемент інтерфейсу що відображає один і більше елементів керування - UITabBarItem. SwiftUI не передбачає альтернатив тож необхідно обернути UIKit елемент у нову структуру що реалізує протокол UIViewRepresentable для iOS та NSViewRepresentable для macOS. Отримали такі структури:

```
struct TabBar: UIViewRepresentable {

    class TabBarDelegate: NSObject, UITabBarDelegate {
        var selectionHandler: ((Int) -> Void)?
```

```
    func tabBar(_ tabBar: UITabBar, didSelect item: UITabBarItem) {
        selectionHandler?(item.tag)
    }
}

var items: [UITabBarItem]
let onSelectItem: (Int) -> Void

private let handler = TabBarDelegate()

func makeUIView(context: UIViewRepresentableContext<TabBar>) -> UITabBar {
    let tabBar = UITabBar()
    tabBar.items = items
    self.handler.selectionHandler = onSelectItem
    tabBar.delegate = self.handler
    return tabBar
}

func updateUIView(_ uiView: UITabBar, context: UIViewRepresentableContext<TabBar>) {
    uiView.items = items
}}
```

2.4 Висновки по розділу

В даному розділі був описаний процес роботи програмного забезпечення з точки зору різних аспектів. Із точки зору взаємодії з користувачем та з точки зору архітектури ПЗ, організації процесу трансляції структур даних.

Також у цьому розділі визначено вхідні і вихідні дані для програмного забезпечення і описано елементи та структури даних для роботи з ними. Наведено приклади вихідного коду.

Було наведено діаграму класів програмного забезпечення мовою розмітки UML. Також його можна, за необхідності, покращити, додавши підтримку нових

View елементів або додавши нову функціональність чи додаткові налаштування імпорту / експорту.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вступ

Якість ПЗ – це набір характеристик програмного забезпечення, що відносяться до його можливості задовольняти встановлені та очікувані вимоги. Для оцінювання якості програмного продукту його потрібно тестувати.

Тестування – це невід'ємна частина процесу розробки будь-якого програмного забезпечення. Основними характеристиками якості програмного забезпечення є:

– зручність використання – можливість використовувати програму без попереднього вивчення документації, зрозумілість усіх функцій;

– надійність – можливість програмного продукту виконувати покладені на нього задачі протягом усього строку використання;

– ефективність – можливість ПЗ забезпечити необхідний рівень швидкодії при обробці файлів;

– функціональність – визначається можливістю програмного продукту вирішувати покладені на нього задачі;

– портативність – степінь можливості переносу ПЗ із одного оточення у інше.

3.2 Функціональність, що підлягає тестуванню

– можливість імпортувати коректний Xib файл;
– можливість імпортувати коректний Storyboard файл;
– серіалізація валідного Xib файла;
– серіалізація валідного Storyboard файла;
– можливість зберегти результат.

3.3 Методологія проведення тестування

Для тестування програмного забезпечення був використаний фреймворк XCTest. Цей фрефмворк розроблений Apple для проведення Unit тестування програмного забезпечення. Тож було написано набір Unit-тестів для перевірки працездатності різних окремих частин ПЗ як от: Імпорт, Трасляція, Експорт.
Він перевіряє вихідні файли на коректність і перевіряє чи усі елементи транслюються у відповідні SwiftUI елементи.

3.4 Опис процесів тестування

Усі ключові функції програмного забезпечення повинні бути покриті Unit тестами. Программа повинна виконувати свої очікувані функції що наведені у списку вимог до ПЗ. Усі тести повинні показувати позитивний результат.

3.5 Процес тестування

Розглянемо процесс тестування за допомогою наведення необхідних тестів. Протестувати необхідно тест-кейси із таблиць 3.1 – 3.5

Таблиця 3.1 – TC001

Назва	можливість імпортувати коректний Xib файл
Передумови	На вхід подано URL що вказує на коректний Xib файл
Дія	Користувач натискає кнопку підтвердження імпорту
Очікуваний результат	Файл зчитано із диску

Таблиця 3.2 – TC002

Назва	можливість імпортувати коректний Storyboard файл

Передумови	На вхід подано URL що вказує на коректний Storyboard файл
Дія	Користувач натискає кнопку підтвердження імпорту
Очікуваний результат	Файл зчитано із диску

Таблиця 3.3 – ТС003

Назва	серіалізація валідного Xib файла
Передумови	На вхід подано вміст валідного Xib файлу
Дія	Прийшов запит на зчитування
Очікуваний результат	Xib серіалізовано у структуру даних що відображає ієрархію його View

Таблиця 3.4 – ТС004

Назва	серіалізація валідного Storyboard файла
Передумови	На вхід подано вміст валідного Storyboard файлу
Дія	Прийшов запит на зчитування
Очікуваний результат	Storyboard серіалізовано у структуру даних що відображає ієрархію його View

Таблиця 3.5 – ТС005

Назва	можливість зберегти результат
Передумови	Програма зчитала валідний Nib файл

Дія	Користувач обрав директорію призначення і запустив процесс експорту
Очікуваний результат	Вихідні файли збережено у обрану директорію

3.6 Висновок до розділу

В даному розділі були розглянуті основні показники якості та в чому полягає визначення поняття якості ПЗ. Було описано методику що буде використано для тестування розробленого програмного продукту. Було наведено тест-кейси.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для запуску розробленого програмного додатку на комп’ютері необхідно, щоб на ньому була встановлена macOS 10.15 Catalina. Необхідно скопіювати на комп’ютер .app Bundle додатку у папку /Applications. Після цього він зявиться у Launchpad і його можна буде запустити кліком по ярлику.

Серед апаратних вимог окремо можна виділити розмір диску. Оскільки при роботі з Xcode проектами розмір Nib-ів і кількіть використаних класів елементів сильно відрізняється то не можливо точно сказати наперед якого розміру одержимо вихідні дані. Більш детально апаратні вимоги розписані у розділі 1.5.1 цієї роботи.

4.2 Робота з програмним забезпеченням

Детальна інструкція користувача наведена у Керівництві користувача.

ВИСНОВКИ

У даній роботі було розроблено програмне забезпечення для трансляції Nib-файлів існуючих проектів у SwiftUI код для переходу до декларативного UI. Систему написано із використанням MVC шаблону, ООП підходу. Сама утиліта є дуже простою у користуванні. Достатньо обрати файл для імпорту і зазначити куди слід експортувати результат.

Вибір мови програмування був обумовлений тим що для розробки потрібна була мова швидка, проста і потужна, а в ідеалі – широко розповсюдженою при написанні macOS додатків. Вибір стояв між Swift та Objective-C, проте обрано було Swift через більшу зручніть використання.

Результатом даної роботи стала macOS Cocoa програма, що може транслювати Nib-файли (а саме Xib та Storyboard) у SwiftUI код.

Було проведено пошук аналогічних проектів і знайдені аналоги було проаналізовано. При тестуванні аналога було виявлено що він лише імітує роботу і корисну роботу не виконує. Це передусім може будти повязано з об’єктивною складністю задачі у зв’язку з великою кількістю елементів та зі складністю трансляції AutoLayout обмежень. Також причиною цього, я вважаю, могло стати те, що сервіс шляхом розміщення такої сторінки-“заглушки” лише вивчає попит на дану утиліту.

Програмне забезпечення є досить складним за будовою і ціна помилки – отримання некоректного SwiftUI файлу, тож було розроблено систему його тестування. Це полегшить внесення змін до коду проекту адже для перевірки роботи буде достатньо виконати наявні тести.

ПЕРЕЛІК ПОСИЛАНЬ

1) SwiftUI | Apple Developer Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.apple.com/documentation/swiftui;

2) Nib file – Cocoa Core Competencies [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/NibFile.html;

3) Model-View-Controller - Cocoa Core Competencies [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html#//apple_ref/doc/uid/TP40008195-CH32-SW1;

4) Geometry Reader To The Resque – The SwiftUI Lab [Електронний ресурс]. – Режим доступу до ресурсу: https://swiftui-lab.com/geometryreader-to-the-rescue/;

5) XMLParser – Foundation | Apple Developer Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.apple.com/documentation/foundation/xmlparser;

6) SwiftUI Layout System [Електронний ресурс]. – Режим доступу до ресурсу: https://kean.blog/post/swiftui-layout-system;

7) Auto Layout Guide: Understanding Auto Layout [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutolayoutPG/index.html;

8) BPMN [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/BPMN;

9) Flutter Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://flutter.dev/docs;

10) Codable – Swift Standard Library | Apple Developer Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.apple.com/documentation/swift/codable.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

Програмне забезпечення для трансляції Nib-файлів у SwiftUI код

Технічне завдання

КПІ.ІП-6307.045490.03.91

“ПОГОДЖЕНО”

Керівник проєкту:

О.А. Халус

Нормоконтроль:

К.І. Ліщук

Виконавець:

О.О. Віхляєв

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмне забезпечення для трансляції Nib-файлів у SwiftUI код

Галузь застосування: Студії та індивідуальні розробники що працюють над програмним забезпеченням для систем Apple із використанням Nib-ів;

Наведене технічне завдання поширюється на розробку macOS програмного додатку “Nib to SwiftUI” [<шифр>], котра використовується для отримання SwiftUI декларативного UI коду із XML розмітки вхідних файлів. Користувачами можуть бути усі, хто мають старі проекти що використовують Nib-файли для побудови користувацького інтерфейсу і хочуть перейти до біль зручного підходу – SwiftUI. Для таких розробників ця утиліта зробить перехід простішим і швидшим.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки Nib to SwiftUI є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (НТУУ «КПІ»).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Програмне забезпечення “Конвертер із XIB у SwiftUI” призначене для поліпшення процесів роботи студій та окремих розробників що працюють із додатками для систем компанії Apple і мають задачі розробки користувацьких інтерфейсів.

Метою створення розробки є поліпшення процесу переводу старих проектів від Nib-файлів до SwiftUI.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

Для користувача:

– спрощення і пришвидшення процесу переходу;

– зменшення стилістичних помилок при переході до нового UI.

Вимоги до організації вхідних даних

Вхідними даними є Nib файли із користувацьким інтерфейсом (Xib або Storyboard).

4.1.2 Вимоги до організації вихідних даних

Вихідними даними є SwiftUI файли із кодом що представляє аналогічний UI до того що був описаний у Nib-файлі.

Розробка була створена виключно під платформу macOS, проте її можливо портувати і на інші системи. Щоправда користі від цього не багато – задачі трансляції Nib у SwiftUI можуть виникати лише у iOS/macOS розробників що користуються виключно системою macOS.

4.1.3 Додаткові вимоги

Не передбачені.

4.2 Вимоги до надійності

Передбачити контроль вхідної інформації. (імпортувати лише файли доступних для обробки форматів)

Забезпечити достатній об’єм вільного дискового сховища протягом експорту результату трансляції.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96

Не висуваються.

4.3.2 Обслуговування

Не потребує додаткового обслуговування

4.3.3 Обслуговуючий персонал

Не потребує додатковий обслуговуючий персонал.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на усіх Apple комп‘ютерах що підтримують macOS 10.15 Catalina.

Мінімальна конфігурація технічних засобів:

4.4.1 Тип процесору

Intel x86-64, сумісний із macOS Catalina

4.4.2 Об‘єм ОЗП

Кількість оперативної пам’яті достатня для нормальної роботи ОС (не менше 4 ГБ).

4.4.3 Об’єм жорсткого диску

Вільне місце на жорсткому диску (зазвичай не менше 20 МБ)

4.5 Вимоги до інформаційної та програмної сумісності

– операційна система macOS 10.15 Catalina і вище;

– вхідними даними повинні бути Nib файли коректної структури, без помилок;

– результати повинні бути представлені у вигляді SwiftUI файлів що зберігаються у директорію обрану користувачем;

– програмне забезпечення повинно надавати можливість налаштування платформ для яких слід генерувати SwiftUI код.

Утиліта була розроблена за допомогою багатопарадигменної мови програмування Swift та фреймворку Cocoa.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати установочну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Попередній склад програмної документації:

5.1 Супроводжувальна документація

5.1.1 Пояснювальна записка;

5.1.2 Керівництво користувача;

5.1.3 Програма та методика тестування.

5.2 Довідникова документація:

5.2.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі;

5.2.2 Програмне забезпечення повинно мати вбудовану довідку.

5.3 Графічна документація:

5.3.1 Схема структурна варіантів використання;

5.3.2 Схема структурна класів програмного забезпечення;

5.3.3 Креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк,	Звітність
1.	Вивчення літератури за тематикою проекту	*19.03.2020*	
2.	Розробка технічного завдання	*26.03.2020*	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	*02.04.2020*	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	*23.04.2020*	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму …)
5.	Програмна реалізація програмного забезпечення	*30.04.2020*	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	*07.05.2020*	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	*21.05.2020*	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проекту	*14.05.2020*	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	*21.05.2020*	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ТРАНСЛЯЦІЇ NIB-ФАЙЛІВ У SWIFTUI КОД

Опис програми

КПІ.ІП-6307.045490.06.13

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.А. Халус

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____О.О. Віхляєв

Київ – 2020 року

Тексти програмного коду

Nib to SwiftUI

(Найменування програми (документа))

DVD-R

(Вид носія даних)

14 арк, 681 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2020

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>CFBundleDevelopmentRegion</key>
	<string>$(DEVELOPMENT_LANGUAGE)</string>
	<key>CFBundleExecutable</key>
	<string>$(EXECUTABLE_NAME)</string>
	<key>CFBundleIconFile</key>
	<string></string>
	<key>CFBundleIdentifier</key>
	<string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>
	<key>CFBundleInfoDictionaryVersion</key>
	<string>6.0</string>
	<key>CFBundleName</key>
	<string>$(PRODUCT_NAME)</string>
	<key>CFBundlePackageType</key>
	<string>$(PRODUCT_BUNDLE_PACKAGE_TYPE)</string>
	<key>CFBundleShortVersionString</key>
	<string>1.0</string>
	<key>CFBundleVersion</key>
	<string>1</string>
	<key>LSMinimumSystemVersion</key>
	<string>$(MACOSX_DEPLOYMENT_TARGET)</string>
	<key>NSHumanReadableCopyright</key>
	<string>Copyright © 2019 Alex. All rights reserved.</string>
	<key>NSMainStoryboardFile</key>
	<string>Main</string>
	<key>NSPrincipalClass</key>
	<string>NSApplication</string>
	<key>NSSupportsAutomaticTermination</key>
	<true/>
	<key>NSSupportsSuddenTermination</key>
	<true/>
</dict>
</plist>
```

```
//
//  AppDelegate.swift
//  XIB to SwiftUI
```

```
//
//  Created by Alex on 10.10.2019.
//  Copyright © 2019 Alex. All rights reserved.
//

import Cocoa
import SwiftUI

@NSApplicationMain
class AppDelegate: NSObject, NSApplicationDelegate {

    var window: NSWindow!

    func applicationDidFinishLaunching(_ aNotification: Notification) {
        // Create the SwiftUI view that provides the window contents.
        let contentView = ImportExportFileView()

        // Create the window and set the content view.
        window = NSWindow(
            contentRect: NSRect(x: 0, y: 0, width: 480, height: 300),
            styleMask: [.titled, .closable, .miniaturizable, .resizable,
.fullSizeContentView],
            backing: .buffered, defer: false)
        window.center()
        window.setFrameAutosaveName("Main Window")
        window.contentView = NSHostingView(rootView: contentView)
        window.makeKeyAndOrderFront(nil)
    }

    func applicationWillTerminate(_ aNotification: Notification) {
        // Insert code here to tear down your application
    }

}




//
//  ImportExportFileView.swift
//  XIB to SwiftUI
//
//  Created by Alex on 10.10.2019.
//  Copyright © 2019 Alex. All rights reserved.
```

```
//

import SwiftUI
import AppKit

struct ImportExportFileView: View {

    @State var importFileURL: URL?
    @State var exportFileURL: URL?

    @State var iOS_enabled: Bool = true
    @State var macOS_enabled: Bool = false

    var body: some View {
        VStack(spacing: 25) {
            Text("Select Nib file to translate it to SwiftUI code")

            if importFileURL != nil {
                Text("Selected " + importFileURL!.lastPathComponent)
                Button(action: {
                    self.translateXibFileToSwiftUI(iOS: self.iOS_enabled, macOS:
self.macOS_enabled)
                }, label: {
                    Text("Translate to SwiftUI")
                })
                GroupBox(label: Text("Target Platforms")) {
                    VStack {
                        Toggle("iOS", isOn: $iOS_enabled)
                        Toggle("macOS", isOn: $macOS_enabled)
                    }
                }
            } else {
                Button(action: {
                    self.showImportFileOpenPanel()
                }, label: {
                    Text("Import Nib file")
                })
            }

        }.padding(50)
    }

    func showImportFileOpenPanel() {
        guard let keyWindow = NSApp.keyWindow else { return }
```

```
        let panel = NSOpenPanel()
        panel.allowedFileTypes = ["xib", "storyboard"]
        panel.beginSheetModal(for: keyWindow) { (result) in
            if result == .OK {
                self.importFileURL = panel.url
            }
        }
    }

    func showExportFileOpenPanel(completion: @escaping () -> Void) {
        guard let keyWindow = NSApp.keyWindow else { return }
        let panel = NSOpenPanel()
        panel.canChooseDirectories = true
        panel.canChooseFiles = false
        panel.beginSheetModal(for: keyWindow) { (result) in
            if result == .OK {
                self.exportFileURL = panel.url
                completion()
            }
        }
    }

    func translateXibFileToSwiftUI(iOS: Bool, macOS: Bool) {
        guard let xibUrl = importFileURL else { return }
        let reader = AVNibReader()
        do {
            try reader.read(from: xibUrl)
            showExportFileOpenPanel {
                guard let swiftUiUrl = self.exportFileURL else { return }
                do {
                    try reader.write(to: swiftUiUrl)
                } catch {
                    guard let keyWindow = NSApp.keyWindow else { return }
                    let alert = NSAlert()
                    alert.messageText = "Error exporting file"
                    alert.informativeText = error.localizedDescription
                    alert.beginSheetModal(for: keyWindow) { (result) in }
                }
            }
        } catch {
            guard let keyWindow = NSApp.keyWindow else { return }
            let alert = NSAlert()
            alert.messageText = "Error importing file"
            alert.informativeText = error.localizedDescription
```

```
            alert.beginSheetModal(for: keyWindow) { (result) in }
        }
    }

}


struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ImportExportFileView()
    }
}
```

```
//
//  AVNibReader.swift
//  XIB to SwiftUI
//
//  Created by Alex on 10.10.2019.
//  Copyright © 2019 Alex. All rights reserved.
//

import AppKit
import IBDecodable

// MARK: - Element Classes

let kUIView = "UIView"
let kUILabel = "UILabel"
let kUIButton = "UIButton"
let kUISegmentedControl = "UISegmentedControl"
let kUITextField = "UITextField"
let kUISlider = "UISlider"
let kUISwitch = "UISwitch"
let kUIActivityindicatorView = "UIActivityindicatorView"
let kUIProgressView = "UIProgressView"
let kUIStepper = "UIStepper"
let kUIStackView = "UIStackView"
let kUITableView = "UITableView"
let kUITableViewContentView = "UITableViewContentView"
let kUIImageView = "UIImageView"
```

```
let kUIToolbar = "UIToolbar"
let kUITabBar = "UITabBar"

// MARK: - XIB Reader

enum AVNibReaderError: Error {
    case noRootView
    case noScenes
    case cantGenerateSwiftUIData
    case unsupportedFileType
}

extension AVNibReaderError {

    var localizedDescription: String {
        switch self {
        case .noRootView:
            return "Can't get root view"
        case .noScenes:
            return "Can't get any scenes"
        case .cantGenerateSwiftUIData:
            return "Can't get SwiftUI view data"
        case .unsupportedFileType:
            return "Only .xib and .storyboard files are supported"
        }
    }

}

enum AVContent {
    case xib(AVRootView)
    case storyboard([AVScene])
}

class AVNibReader {

    private var content: AVContent?

    // MARK: - Read & Write

    func read(from url: URL) throws {
        switch url.pathExtension {
        case "xib":
            try read(xibUrl: url)
```

```
            case "storyboard":
                try read(storyboardUrl: url)
            default:
                throw AVNibReaderError.unsupportedFileType
            }
        }

        private func read(xibUrl: URL) throws {
            let xib = try XibFile(url: xibUrl)
            guard let view = xib.document.views?.first?.view else {
                throw AVNibReaderError.noRootView
            }

            let rootView = rootViewFrom(view: view)

            content = .xib(rootView)
        }

        private func read(storyboardUrl: URL) throws {
            let storyboard = try StoryboardFile(url: storyboardUrl)

            let scenes = (storyboard.document.scenes ?? []).compactMap({ (scene) ->
AVScene? in
                guard let view = scene.viewController?.viewController.rootView else {
                    return nil
                }

                let rootView = rootViewFrom(view: view)

                let customViews = (scene.customViews ?? []).map { (view) -> AVRootView
in
                    return rootViewFrom(view: view.view)
                }

                return AVScene(additionalViews: customViews, rootView: rootView)
            })

            guard !scenes.isEmpty else {
                throw AVNibReaderError.noScenes
            }

            content = .storyboard(scenes)
        }
```

```
    func write(to outputFolderUrl: URL) throws {
        guard let content = content else {
            return
        }

        switch content {
        case .xib(let rootView):
            guard let swiftUiViewData = rootView.stringRepresentation.data(using: .utf8)
else {
                throw AVNibReaderError.cantGenerateSwiftUIData
            }
            try swiftUiViewData.write(to: outputFolderUrl)
        case .storyboard(let scenes):
            func saveScene(_ scene: AVScene, in folderUrl: URL) {
                // Save Root view
                let rootSwiftUiViewData =
scene.rootView.stringRepresentation.data(using: .utf8)
                try? rootSwiftUiViewData?.write(to:
folderUrl.appendingPathComponent("SceneRootView.swift"))
                // Save Additional views
                let additionalSwiftUiViewsData = scene.additionalViews.map({
$0.stringRepresentation.data(using: .utf8) })
                let additionalViewsFolderUrl =
folderUrl.appendingPathComponent("AdditionalViews", isDirectory: true)
                try? FileManager.default.createDirectory(at: additionalViewsFolderUrl,
withIntermediateDirectories: true, attributes: nil)
                for (index, viewData) in additionalSwiftUiViewsData.enumerated() {
                    try? viewData?.write(to:
additionalViewsFolderUrl.appendingPathComponent("AdditionalView_\(index).swif
t"))
                }
                // Save Custom OS classes

            }
            for (index, scene) in scenes.enumerated() {
                let sceneUrl = outputFolderUrl.appendingPathComponent("Scene_\(index
+ 1)", isDirectory: true)
                try? FileManager.default.createDirectory(at: sceneUrl,
withIntermediateDirectories: true, attributes: nil)
                saveScene(scene, in: sceneUrl)
            }
        }
    }
```

```
// MARK: - Parsing

func rootViewFrom(view: ViewProtocol) -> AVRootView {
    let viewClass = (view.customClass ?? view.elementClass)
    let wrappedView = translate(view: view)!
    return AVRootView(className: viewClass, view: wrappedView)
}

func translate(view: ViewProtocol) -> AVBaseView? {
    switch view.elementClass {
    case kUIView:
        return translateView(from: view)
    case kUILabel:
        return translateLabel(from: view)
    case kUIButton:
        return translateButton(from: view)
    case kUISegmentedControl:
        return translateSegmentedControl(from: view)
    case kUITextField:
        return translateTextField(from: view)
    case kUISlider:
        return translateSlider(from: view)
    case kUISwitch:
        return translateSwitch(from: view)
    case kUIActivityindicatorView:
        return translateActivityIndicatorView(from: view)
    case kUIProgressView:
        return translateProgressView(from: view)
    case kUIStepper:
        return translateStepper(from: view)
    case kUIStackView:
        return translateStackView(from: view)
    case kUITableView:
        return translateTableView(from: view)
    case kUITableViewContentView:
        return translateTableViewContentView(from: view)
    case kUIImageView:
        return translateImageView(from: view)
    case kUIToolbar:
        return translateToolbar(from: view)
    case kUITabBar:
        return translateTabBar(from: view)
    default:
        return nil
```

```
        }
    }

    // MARK: - Element translation

    func translateView(from view: ViewProtocol) -> AVView? {
        // if in line - return stack view

        // if complex - geometry reader

        let viewBackgroundColor = view.backgroundColor?.sRGB.map{
AVColor(color: $0) } ?? .clear
        let viewSubviews = (view.subviews?.map({ $0.view }) ?? []).compactMap({
(view) -> AVBaseView? in
            return translate(view: view)
        })
        return AVView(backgroundColor: viewBackgroundColor, subviews:
viewSubviews)
    }

    func translateLabel(from view: ViewProtocol) -> AVLabel? {
        guard let label = view as? IBDecodable.Label else {
            return nil
        }
        var font: AVFont?
        if let fontDescription = label.fontDescription {
            font = AVFont(font: fontDescription)
        }
        var color: AVColor?
        if let textColor = label.textColor?.sRGB {
            color = AVColor(color: textColor)
        }
        return AVLabel(text: label.text, font: font, textColor: color ?? .black)
    }

    func translateButton(from view: ViewProtocol) -> AVButton? {
        guard let button = view as? IBDecodable.Button else {
            return nil
        }
        let state = button.state?.first
        return AVButton(title: state?.title ?? "", titleColor: nil, imageName: nil)
    }
```

```
    func translateSegmentedControl(from view: ViewProtocol) ->
AVSegmentedControl? {
        guard let segmentedControl = view as? IBDecodable.SegmentedControl else {
            return nil
        }
        let segments = segmentedControl.segments.map { (segment) ->
AVSegmentedControl.Segment in
            return .text(segment.title)
        }
        let selected = segmentedControl.selectedSegmentIndex ?? 0
        return AVSegmentedControl(segments: segments, selectedSegment: selected)
    }

    func translateTextField(from view: ViewProtocol) -> AVTextField? {
        guard let textField = view as? IBDecodable.TextField else {
            return nil
        }
        return AVTextField(text: textField.text)
    }

    func translateSlider(from view: ViewProtocol) -> AVSlider? {
        guard let _ = view as? IBDecodable.Slider else {
            return nil
        }
        return AVSlider(value: 0.5, range: Swift.Range(uncheckedBounds: (0, 1)))
    }

    func translateSwitch(from view: ViewProtocol) -> AVSwitch? {
        guard let _switch = view as? IBDecodable.Switch else {
            return nil
        }
        return AVSwitch(value: _switch.on)
    }

    func translateActivityIndicatorView(from view: ViewProtocol) ->
AVActivityIndicatorView? {
        guard let _ = view as? IBDecodable.ActivityindicatorView else {
            return nil
        }
        return AVActivityIndicatorView(isAnimating: true)
    }

    func translateProgressView(from view: ViewProtocol) -> AVProgressView? {
        guard let _ = view as? IBDecodable.ProgressView else {
```

```
            return nil
        }
        return AVProgressView(progress: 0.5)
    }

    func translateStepper(from view: ViewProtocol) -> AVStepper? {
        guard let _ = view as? IBDecodable.Stepper else {
            return nil
        }
        return AVStepper(value: 1, range: Swift.Range(uncheckedBounds: (0, 10)))
    }

    func translateStackView(from view: ViewProtocol) -> AVStackView? {
        guard let stack = view as? IBDecodable.StackView else {
            return nil
        }
        let axis: AVStackView.Axis = (stack.axis == "vertical") ? .vertical : .horisontal
        let subviews = view.subviews?.map({ $0.view }) ?? []
        return AVStackView(axis: axis, spacing: 0.0, alignment: .fill, distribution: .fill,
subviews: subviews.compactMap({ (view) -> AVBaseView? in
            return translate(view: view)
        }))
    }

    func translateTableView(from view: ViewProtocol) -> AVTableView? {
        guard let tableView = view as? IBDecodable.TableView else {
            return nil
        }
        let contentMode: AVTableView.ContentMode = (tableView.dataMode ==
.prototypes) ? .dynamic : .static
        let subviews = tableView.prototypeCells?.compactMap({ $0.contentView }) ??
[]
        return AVTableView(mode: contentMode, cells: subviews.compactMap({
(view) -> AVBaseView? in
            return translate(view: view)
        }))
    }

    func translateTableViewContentView(from view: ViewProtocol) ->
AVTableViewCellContentView? {
        guard let contentView = view as?
IBDecodable.TableViewCell.TableViewContentView else {
            return nil
        }
```

```
        let subviews = contentView.subviews?.map({ $0.view }).compactMap({ (view)
    -> AVBaseView? in
            return translate(view: view)
        }) ?? []
        return AVTableViewCellContentView(subviews: subviews)
    }

    func translateImageView(from view: ViewProtocol) -> AVImageView? {
        guard let imageView = view as? IBDecodable.ImageView else {
            return nil
        }
        return AVImageView(imageName: imageView.image ?? "")
    }

    func translateToolbar(from view: ViewProtocol) -> AVToolbar? {
        guard let toolbar = view as? IBDecodable.Toolbar else {
            return nil
        }
        let items = toolbar.items?.map { (barButtonItem) -> AVToolbar.Item in
            if let _ = barButtonItem.systemItem {
                return AVToolbar.Item.system
            } else if let text = barButtonItem.title {
                return AVToolbar.Item.title(text)
            } else {
                return AVToolbar.Item.system
            }
        } ?? []
        return AVToolbar(items: items)
    }

    func translateTabBar(from view: ViewProtocol) -> AVTabBar? {
        guard let tabBar = view as? IBDecodable.TabBar else {
            return nil
        }
        let items = tabBar.items?.map { (tabBarItem) -> AVTabBar.Item in
            if let systemName = tabBarItem.systemItem {
                return AVTabBar.Item.system(systemName)
            } else {
                return AVTabBar.Item.custom(tabBarItem.title, nil)
            }
        } ?? []
        return AVTabBar(items: items)
    }
```

```
        // TODO: Other


    }
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

Програмне забезпечення для трансляції Nib-файлів у SwiftUI код

Програма та методика тестування

КПІ.ІП-6307.045490.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.А. Халус

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____О.О. Віхляєв

Київ – 2020 року

ЗМІСТ

5 ОБ’ЄКТ ВИПРОБУВАНЬ

Програмне забезпечення для трансляції Nib-файлів у SwiftUI код. Дане програмне забезпечення побудоване за MVC парадигмою у об’єктно-орієнтованому стилі. Програмний продукт може бути запущеним на macOS 10.15 Catalina і вище.

6 МЕТА ТЕСТУВАННЯ

Під час тестування тестуються наступні функції:

– можливість запуску на усіх версіях macOS 10.15.x Catalina;

– можливість безперешкодно обрати допустимий тип файлу;

– неможливість обирати файли усіх типів окрім допустимих;

– коректна серіалізація коректних вхідних файлів;

– оброблення помилки при некоректних вхідних файлах;

– експортування файлів у обрану директорію;

– коректність сгенерованих SwiftUI структур;

– коректне виконання зачитування текстів;

– коректне функціонування карти з позначеними на ній місцями, де тексти були скановані;

– тестування UI.

7 МЕТОДИ ТЕСТУВАННЯ

Тестування проводиться методом напівпрозорого ящика. Метод намівпрозорого ящика – це тестування внутрішньої структури ПЗ, головних функцій та алгоритмів, із точки зору користувача додатку. Для кожної такої частини функціональності створюються окремі тест-кейси що перевіряють коректність чітко описаної функції.

Тестування проводилося за рядом методів:

– ui тестування;

– юзабіліті тестування – перевіряє зручність додатку у використанні;

– функціональне тестування – перевіряє функціонал застосунку на відповідність поставленим вимогам.

8 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

UI тестування проводиться шляхом перевірки UI додатку на відповідність макетам при відображені на різних пристроях виводу зображення.

Юзабіліті тестування проводиться емпіричним методом.

Функціональне тестування виконане шляхом модульного тестування, тобто тестування кожної атомарної функції ПЗ окремо в штучно створеному середовищі. Для цього використано XCTest framework.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

«Програмне забезпечення для трансляції Nib- файлів у SwiftUI код»

Керівництво користувача

КПІ.ІП-6307.045490.05.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.А. Халус

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____О.О. Віхляєв

Київ – 2020 року

ЗМІСТ

1 ІНСТРУКЦІЯ КОРИСТУВАЧА

1.1 Запуск програмного забезпечення

Для початку роботи з програмним забезпеченням користувачу не потрібно авторизуватися. Усе що потрібно – запустити .app файл додатку на сумісній операційній системі.

1.2 Вхідні параметри

Суть роботи полягає у трансляції Nib файлів у SwiftUI код. Тож як вхід програмне забезпечення приймає Nib файли (Xib або Storyboard). Вибір файлу зображений на Рисунку 1:

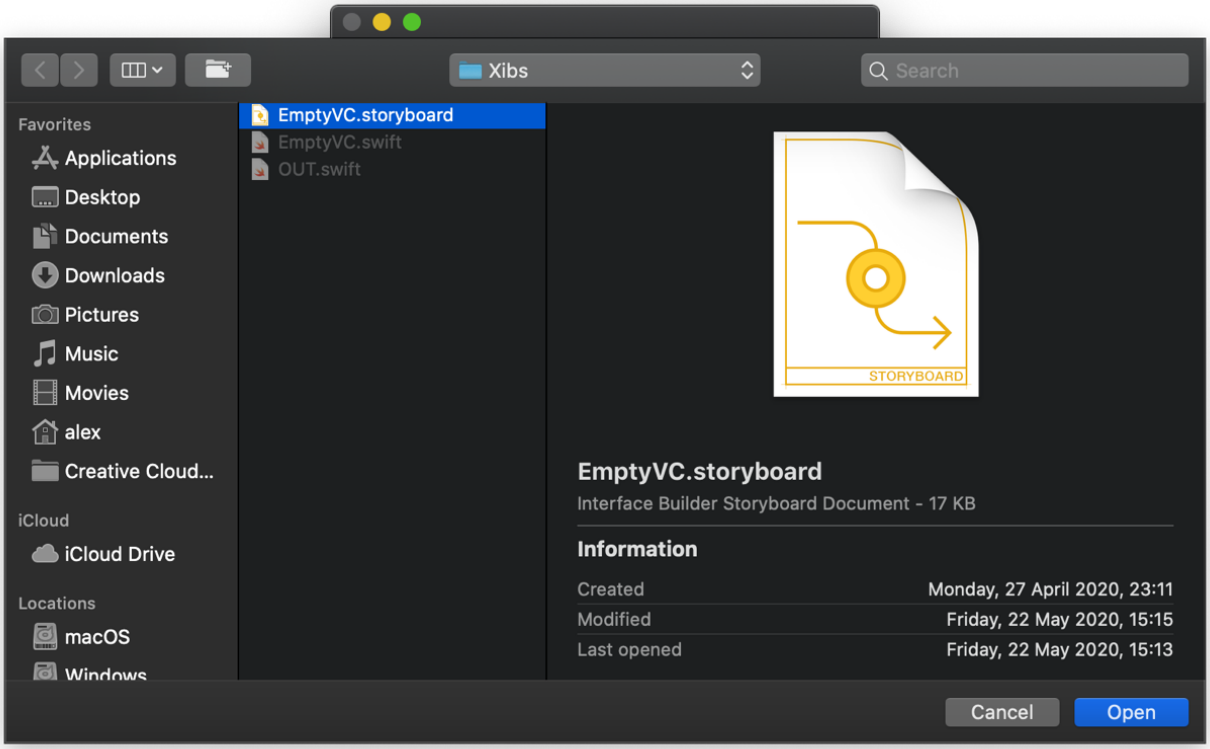


Рисунок 1 – Імпорт Nib файлу

Після цього користувач обирає директорію призначення і програма зберігає результат трансляції:

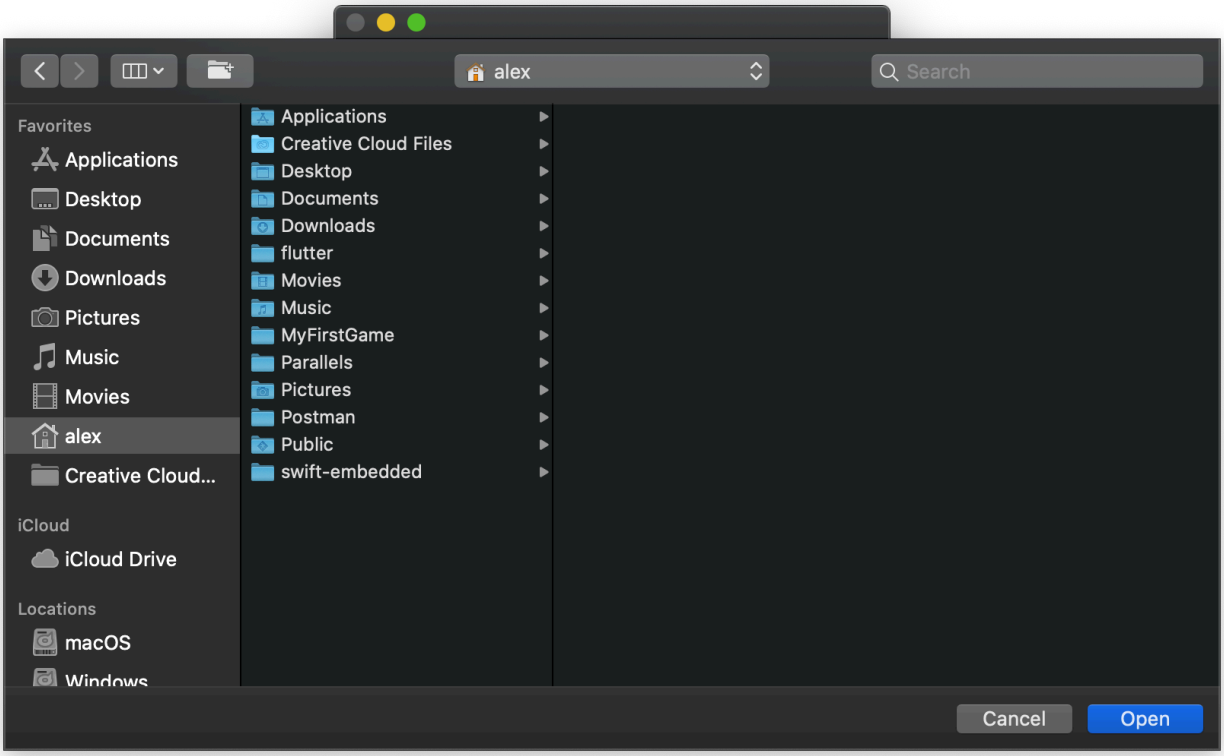


Рисунок 2 – Вибір директорії призначення

1.3 Перегляд детального звіту

Файли результату зберігаються у обрану директорію, при чому для кожної сцени буде створено піддиректорію.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

Програмне забезпечення для трансляції Nib-файлів у SwiftUI код

Графічні матеріали

КПІ.ІП-6307.045490.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.А. Халус

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____О.О. Віхляєв

Київ – 2020 року

КПІ.ІП-6307.045490.07.99

					КПІ.ІП-6307.045490.07.99	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використання	Літера / Маса / Масштаб
Розробив		Віхляєв О.О.				
Перевірив		Халус О.А.				Аркуш / Аркушів
Т. кон.						
					Програмне забезпечення для трансляції Nib-файлів у SwiftUI код	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63
Н. кон.		Ліщук К.І.				
Затвердив		Халус О.А.				

КПІ.ІП-6307.045490.07.99

					КПІ.ІП-6307.045490.07.99		
						Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів програмного забезпечення			
Розробив		Віхляєв О.О.				Аркуш	Аркушів	
Перевірив		Халус О.А.						
Т. кон.								
Н. кон.		Ліщук К.І.			Програмне забезпечення для трансляції Nib-файлів у SwiftUI код	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63		
Затвердив		Халус О.А.						

КПІ.ІП-6307.045490.07.99

					КПІ.ІП-6307.045490.07.99			
						Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм			
Розробив		Віхляєв О.О.				Аркуш	Аркушів	
Перевірив		Халус О.А.						
Т. кон.					Програмне забезпечення для трансляції Nib-файлів у SwiftUI код	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63		
Н. кон.		Ліщук К.І.						
Затвердив		Халус О.А.						